****

****

Jianqi Zhang<jianqiz@usc.edu>, 1052509893

2020/03/05

****

Odd-even transposition sort algorithm is a parallel sorting algorithm. It sorts n elements in n

clocks (n is even), each of which requires n/2 compare-exchange operations. This algorithm

alternates between two phases, called the odd and even phases. Let < a1; a2; … ; an > be the

sequence to be sorted. During the odd phase, elements with odd indices are compared with

their right neighbors, and if they are out of sequence they are exchanged; thus, the pairs

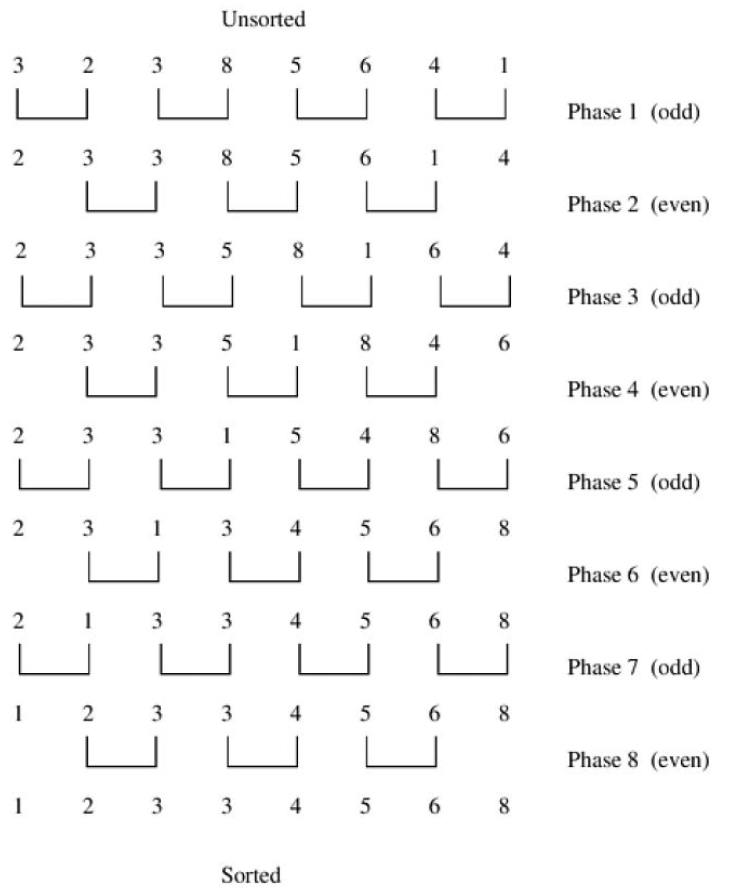
(a1; a2); (a3; a4); …; (an-1; an) are compare-exchanged (assuming n is even). Similarly, during

the even phase, elements with even indices are compared with their right neighbors, and if

they are out of sequence they are exchanged; thus, the pairs (a2; a3); (a4; a5); :::; (an-2; an-1)

are compare-exchanged. After n phases of odd-even exchanges, the sequence is sorted. An

example sorting instance is shown in Figure 1.

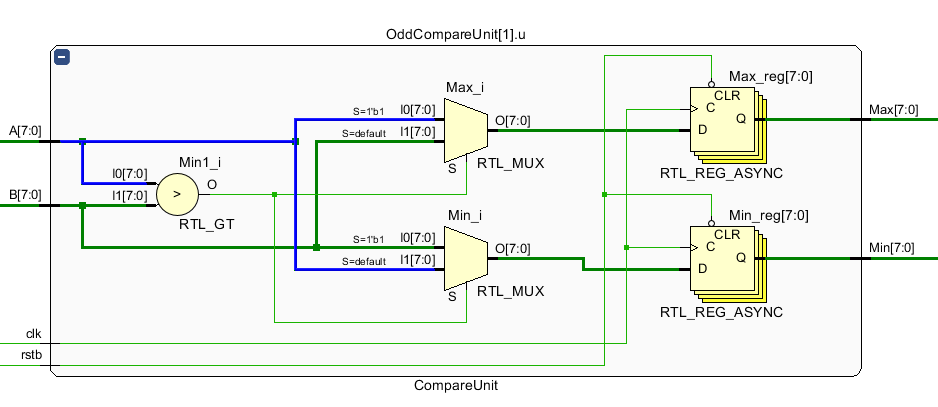
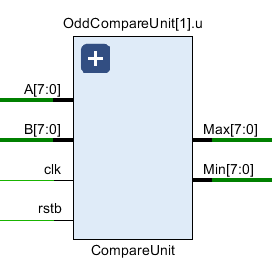


* 

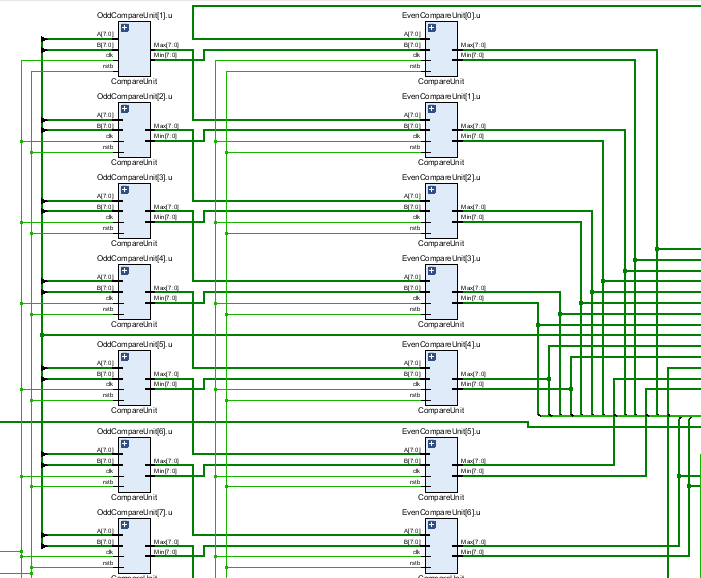
1. Using Verilog, implement odd-even transposition circuit, which takes n, 8 bit inputs and sort them.
2. For a 16 elements write a test bench and verify the waveforms.
3. Elaborate the design and include all the schematics' screenshots of the modules in the report.
4. Synthesis the design and include the screenshots.
5. Generate Resource and timing estimations and include them in the report.
6. Redo part 3, 4, 5 for 32, 64, 128.

**Schematics’ screenshots** are shown below:

1. **OddCompareUnit** is designed to compare two 8-bit number A and B, and output the maximum one and minimum one between them.

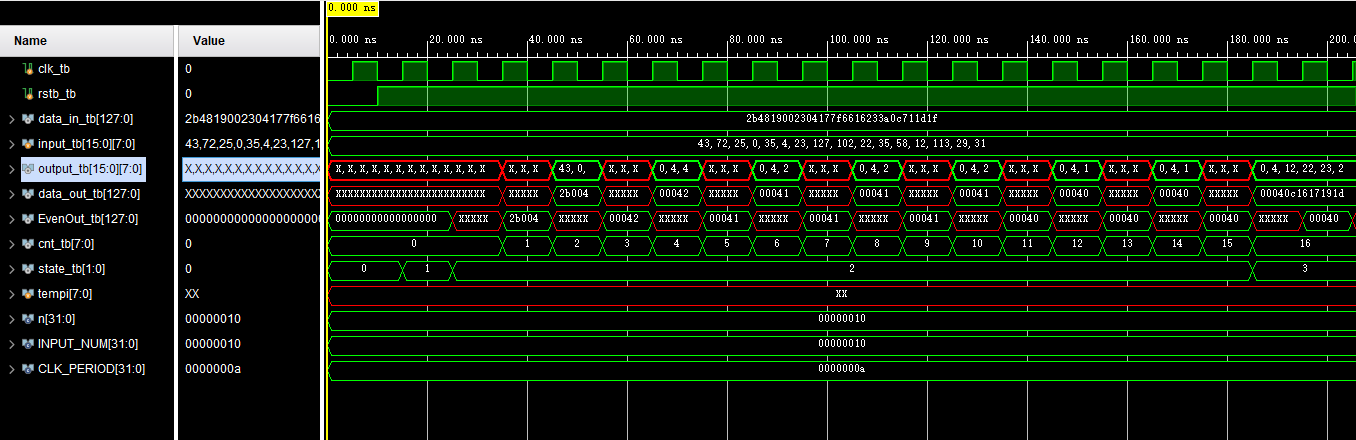


1. **OddevenSort** is the top module of the design. The main part of this module is an array of compare unit, which consist of 8 odd compare units and 7 even compare units.

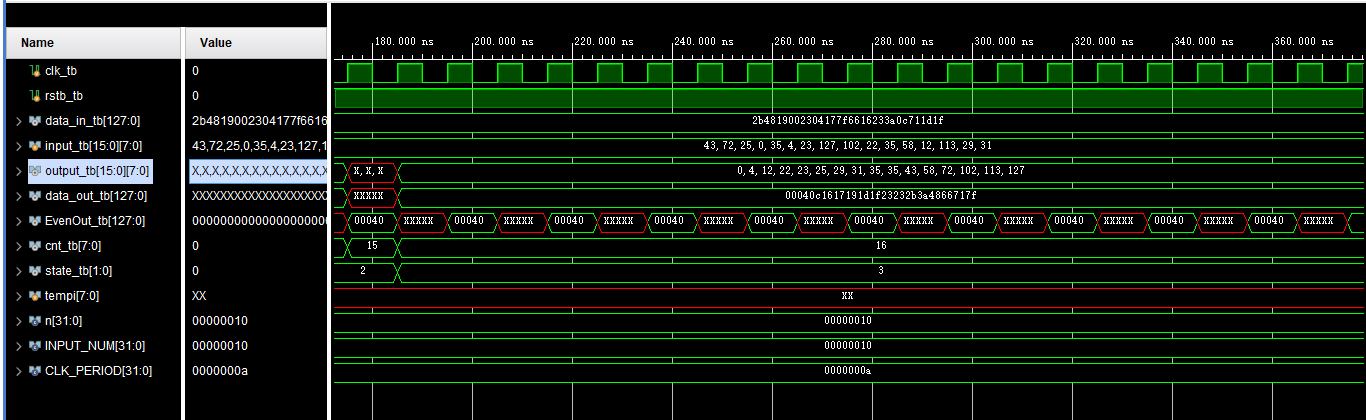


**The total** **schematic is shown in the Appendix.**

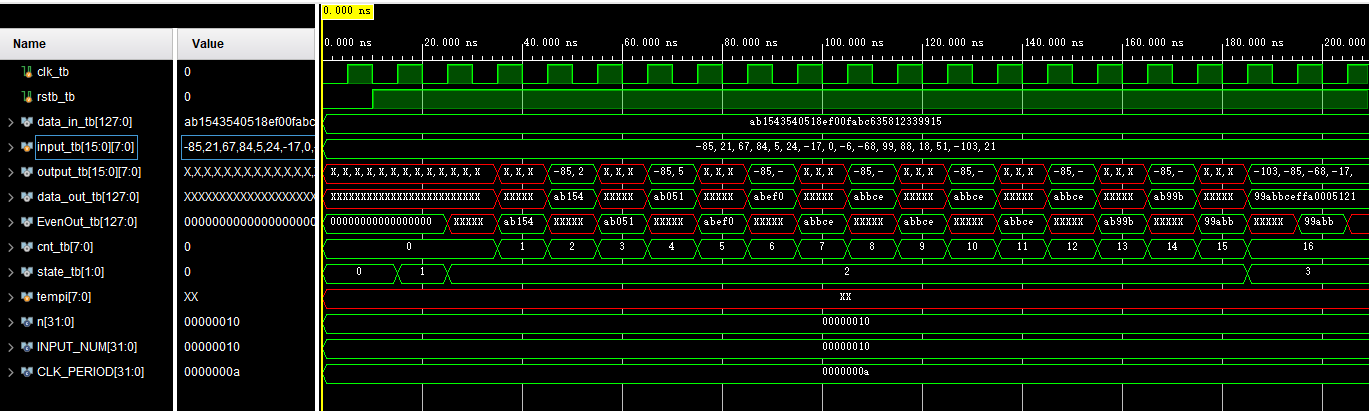
For a 16 elements test bench, we give 16 8-bit numbers as input, and after 2\*16 = 32 clocks, the module will output sorted results. The waveform is shown as followed:

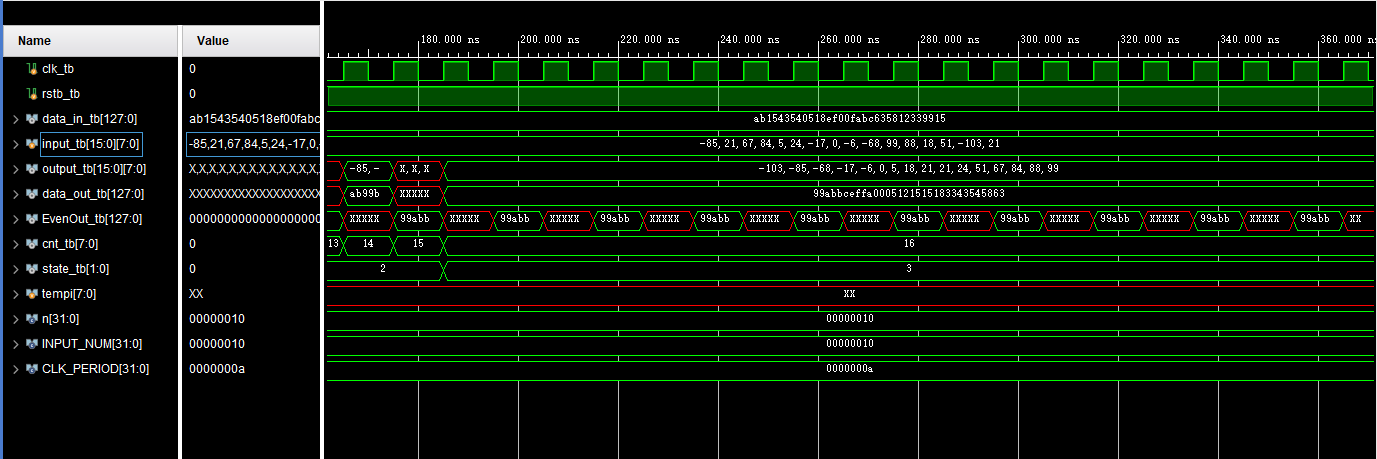


We can see input\_tb includes 16 8-bit numbers, after 16 clocks, output\_tb gives the sorted number array.



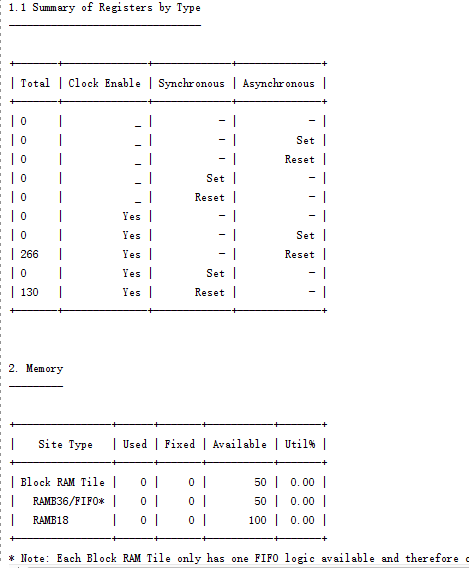
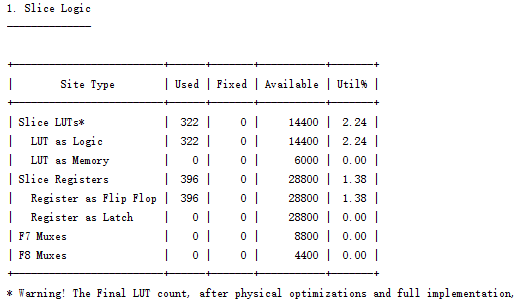
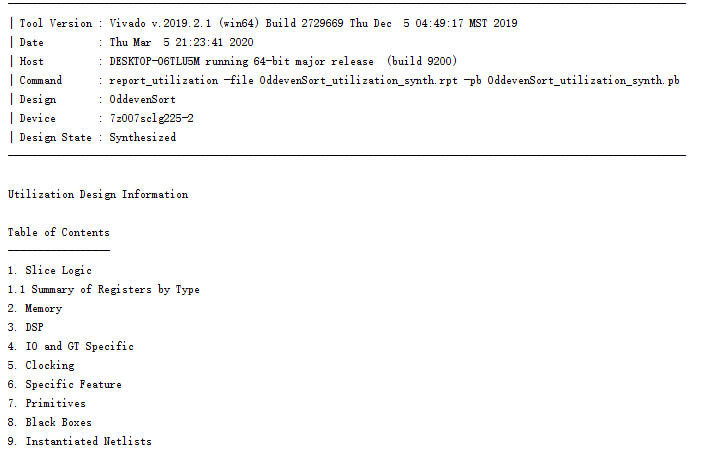
Then, we give another sets of input.

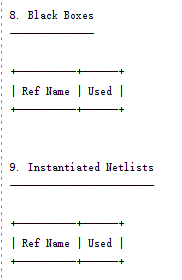
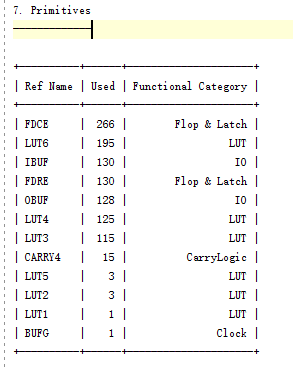
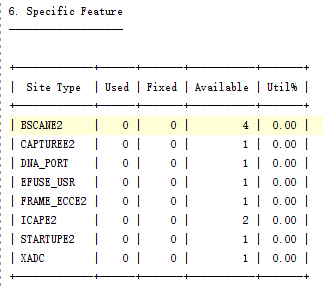
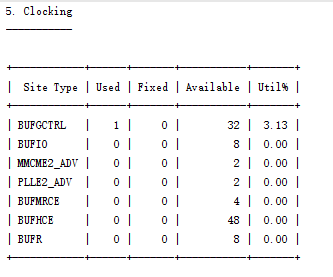
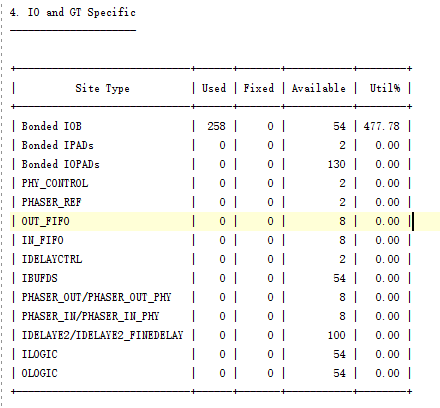
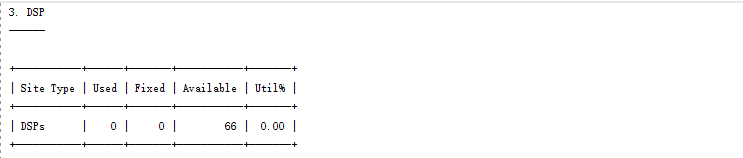




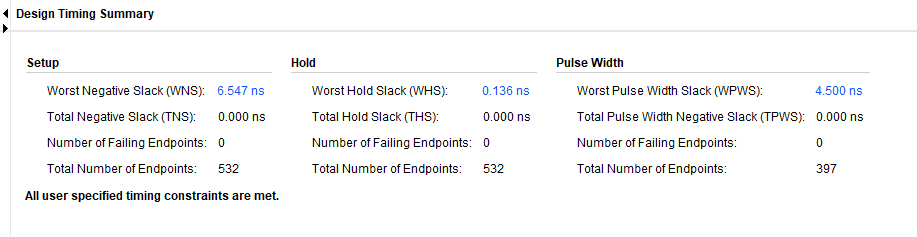
**The full synthesis schematic please see in the Appendix.**

**Resource Estimations:**

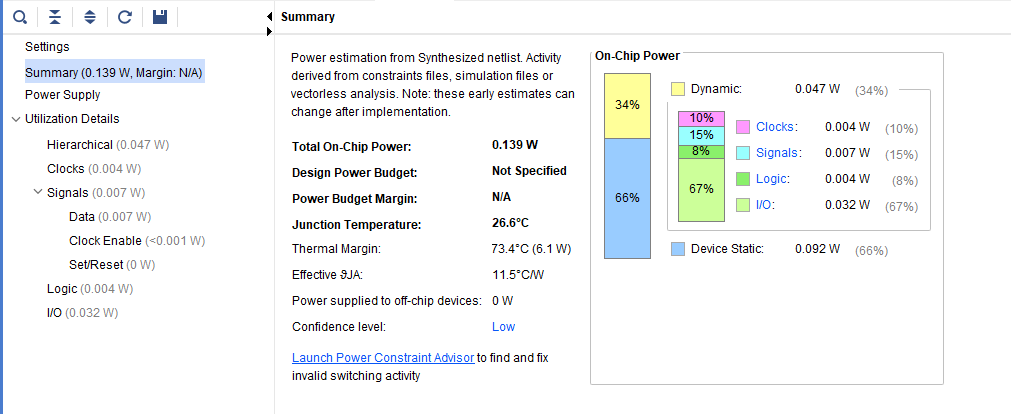




**Timing Estimation:**



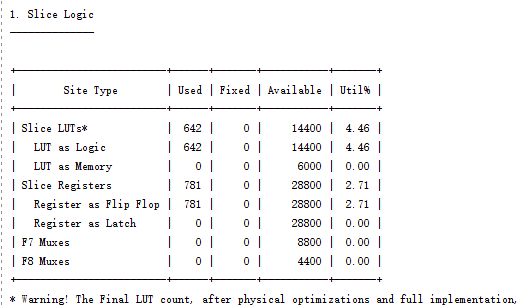
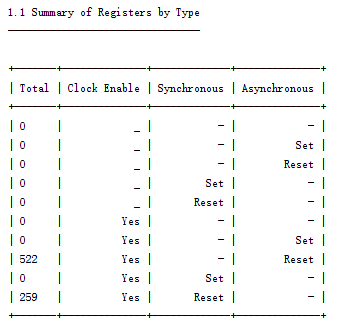
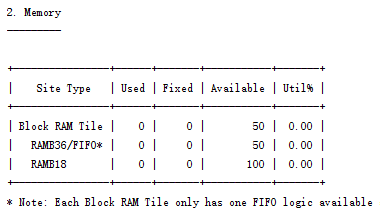
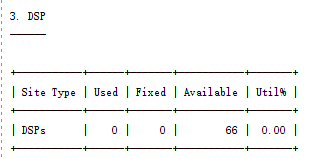
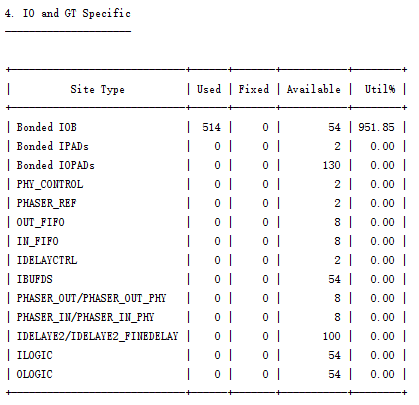
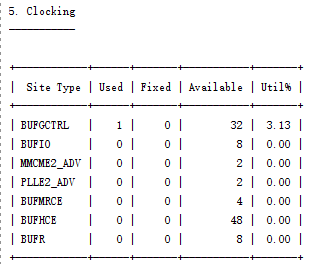
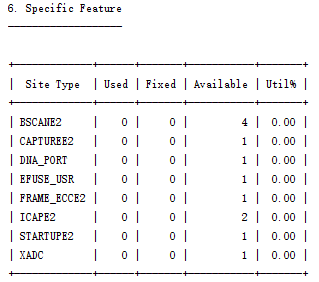
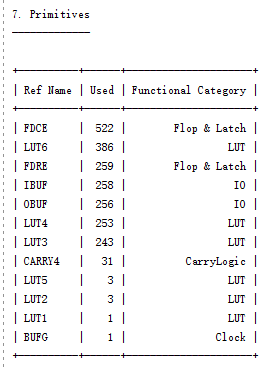
**Power Estimation:**



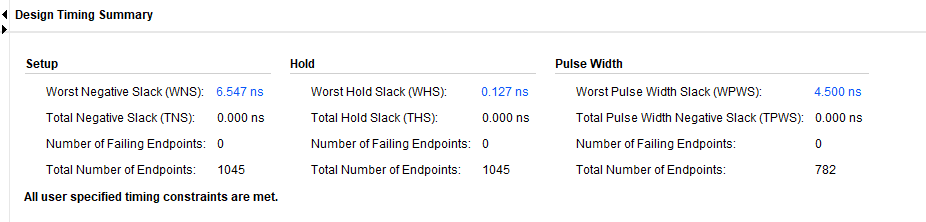
**Now redo for 32 inputs.**

**The full synthesis schematic please see in the Appendix.**

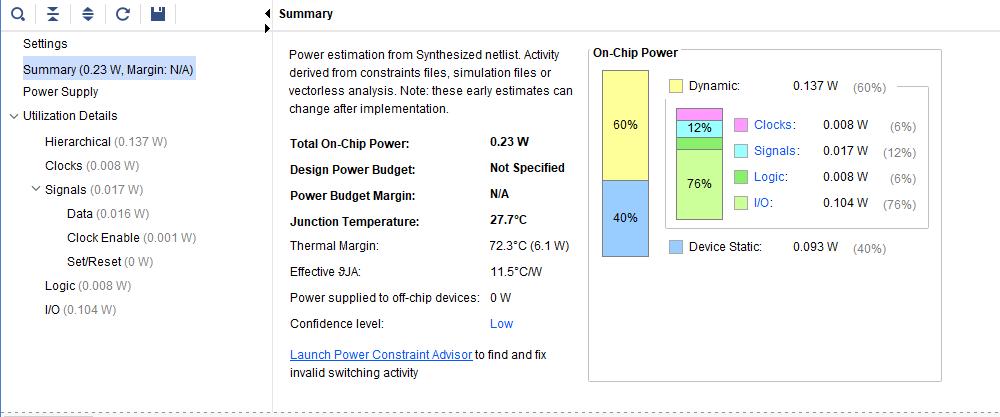
**Resource Estimation:**

**Timing Estimation:**



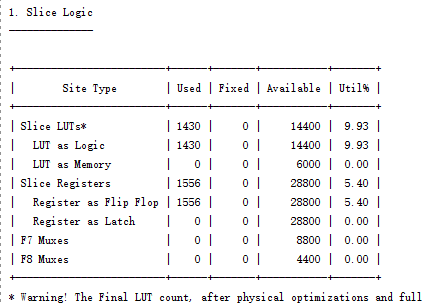
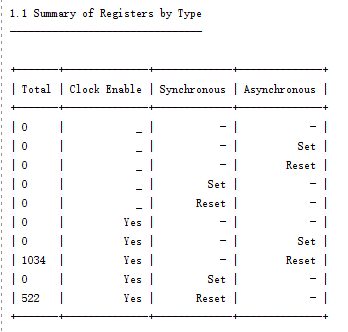
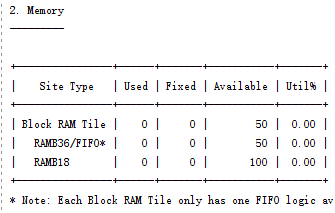
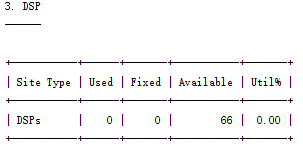
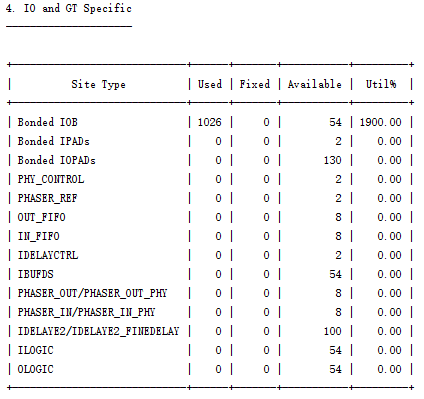
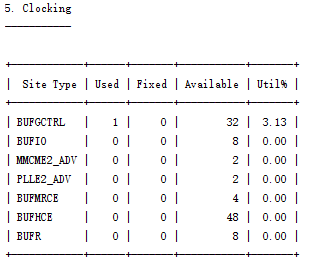
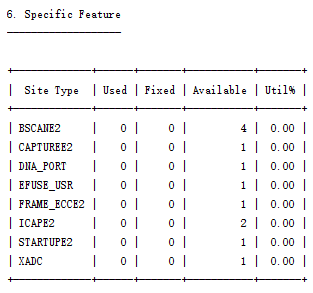
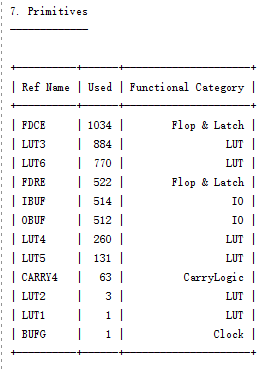
**Power Estimation:**



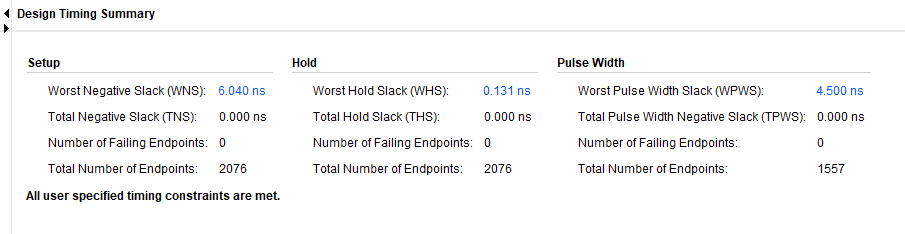
**Now redo for 64 inputs.**

**The full synthesis schematic please see in the Appendix.**

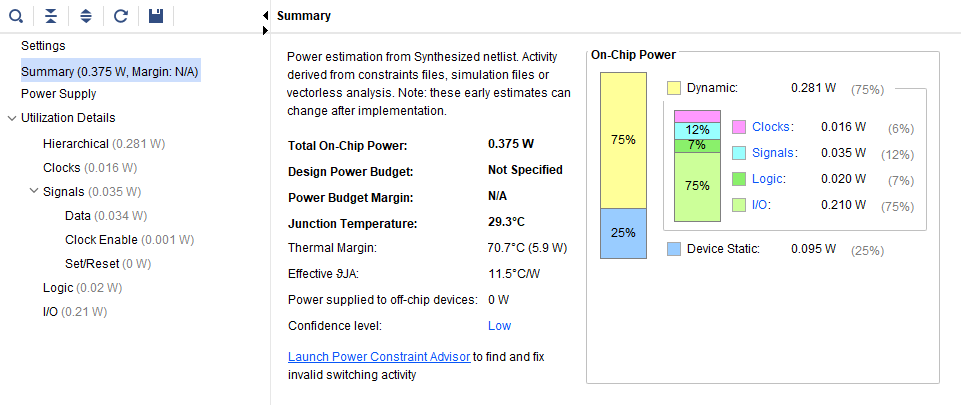
**Resource Estimation:**

**Timing Estimation:**



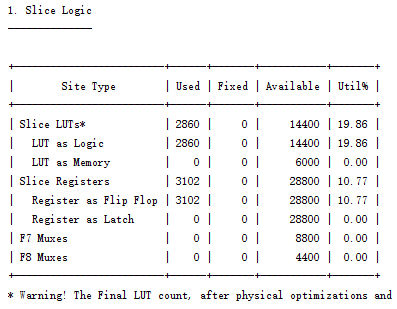
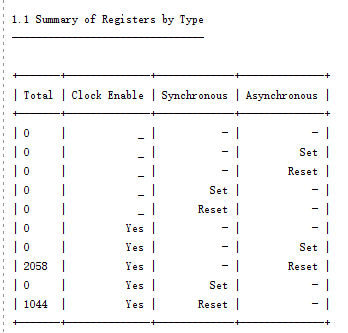
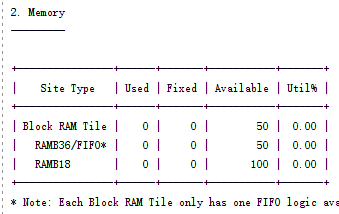
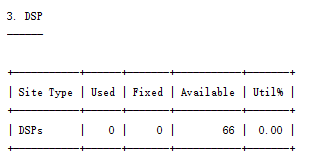
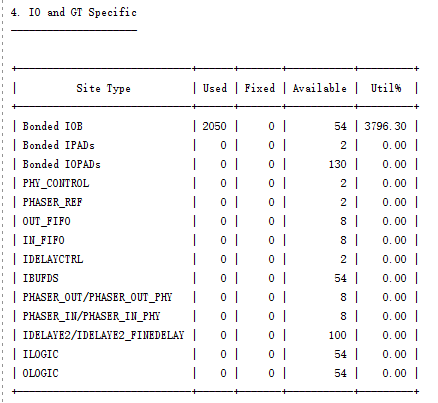
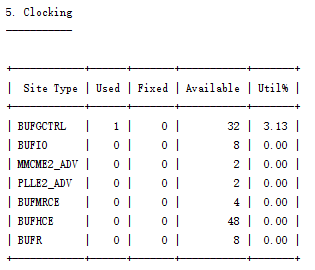
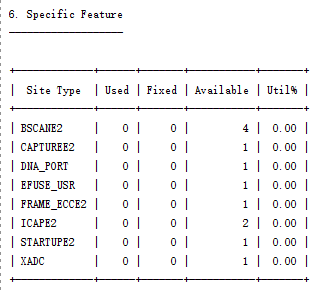
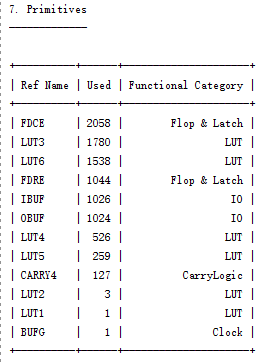
**Power Estimation:**



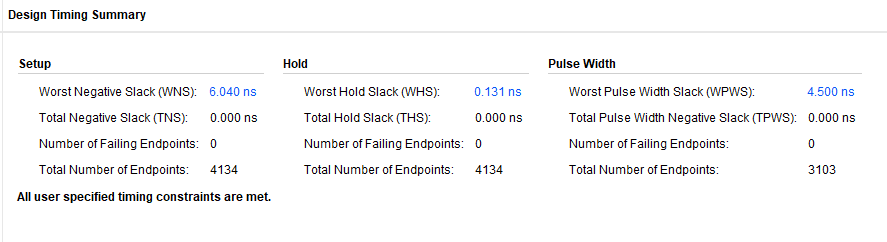
**Now redo for 128 inputs.**

**The full synthesis schematic please see in the Appendix.**

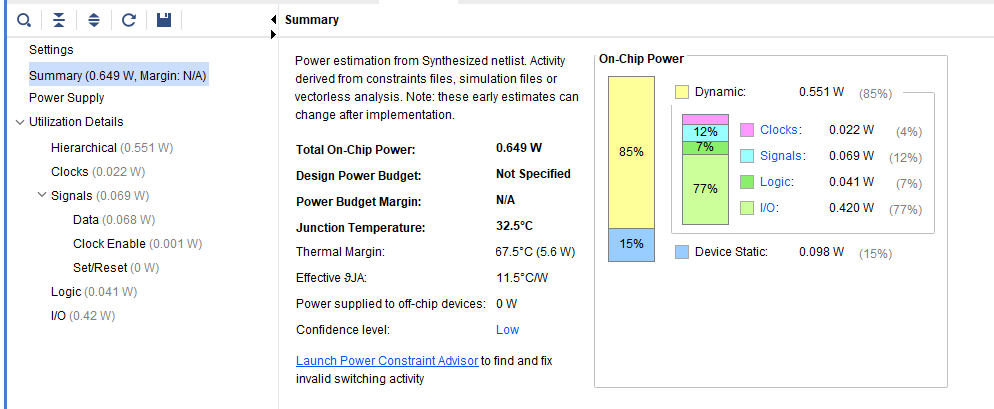
**Resource Estimation:**

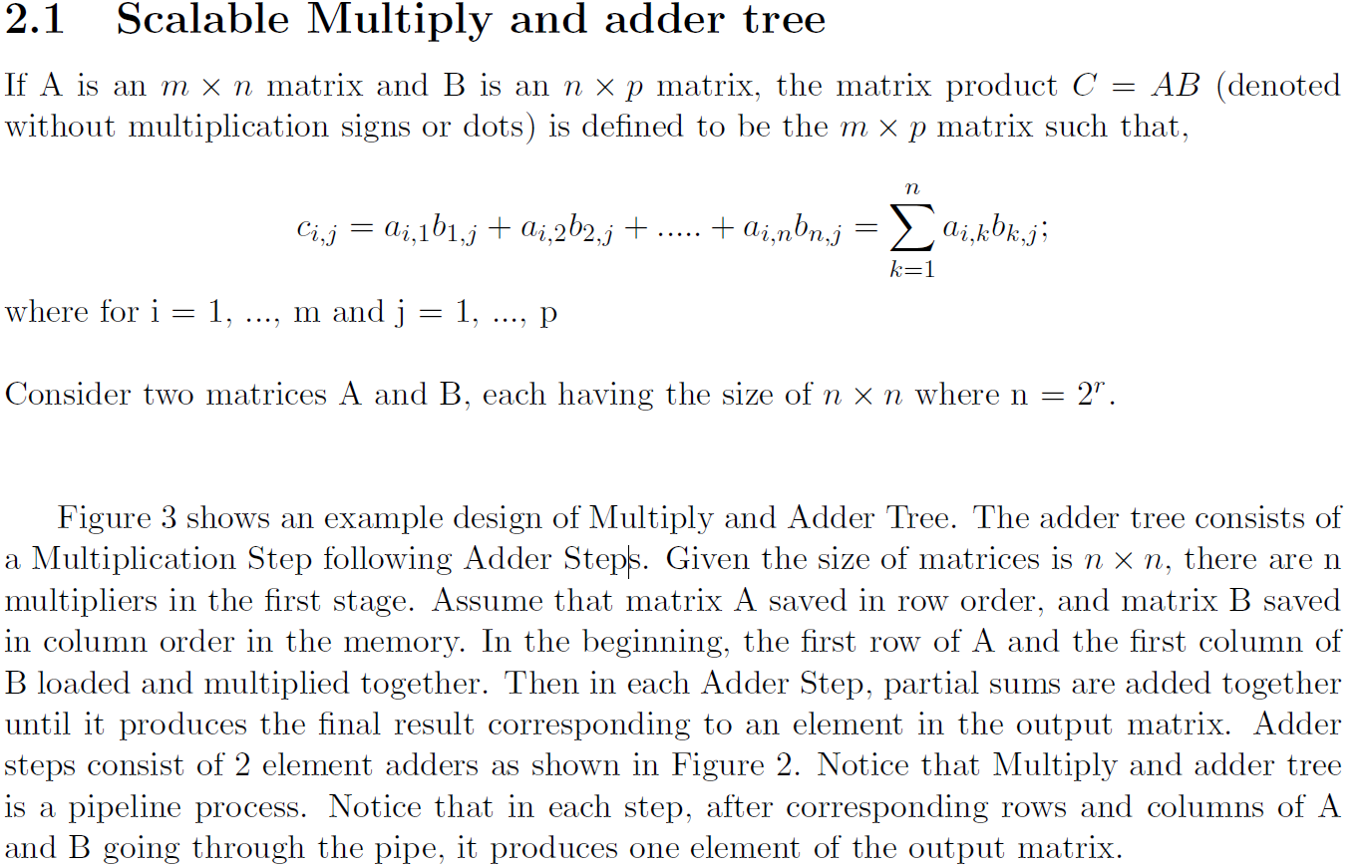
**Timing Estimation:**



**Power Estimation:**



****



**2.1.1 Design Problems**

Consider simple Multiply and Adder Tree design with n element multiplication,

1. How many Multiply units needed for the entire design?

2. Consider an adder stage, how many adder modules needed for that stage (Assume

multiplication stage as stage 0)?

3. If all the inputs to the design represented using k bits, how many bits are needed to

represent the final result of the Multiply and Adder Tree?

4. How may Adder modules need for the entire multiply and adder tree design?

5. How many clock cycles need to produce the first output element in the adder tree?

6. How many clock cycles need to multiply two n×n matrices?

**2.1.2 Implementation**

1. Write a testbench which provides two 4X4 matrices to the design and takes output final 4X4 resultant matrix.

2. Simulate the design using the simulator and include the waveform in the report (Clearly indicate the locations where final outputs produced)

3. Elaborate the design and include all the schematics' screenshots of the modules (MulandAddTree, adder, and multiply) in the report.

4. Synthesis the design and include the screenshots like part 3.

5. Generate Resource and timing estimations and include them in the report.

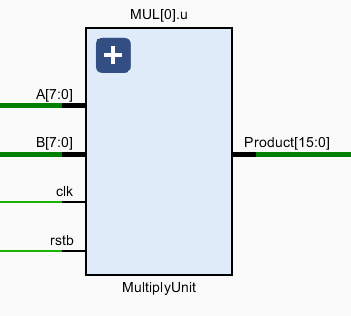
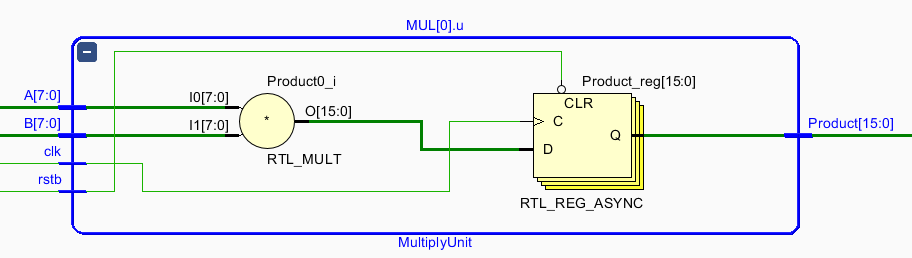
6. Generate power estimation reports and include them in the report.

7. How many of parallel *MulandAddTrees* can be implemented in this FPGA (Provide resource utilization reports with parallel *MulandAddTres*)?

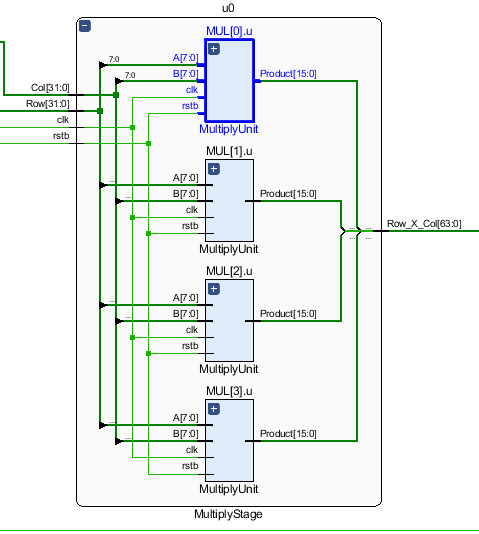
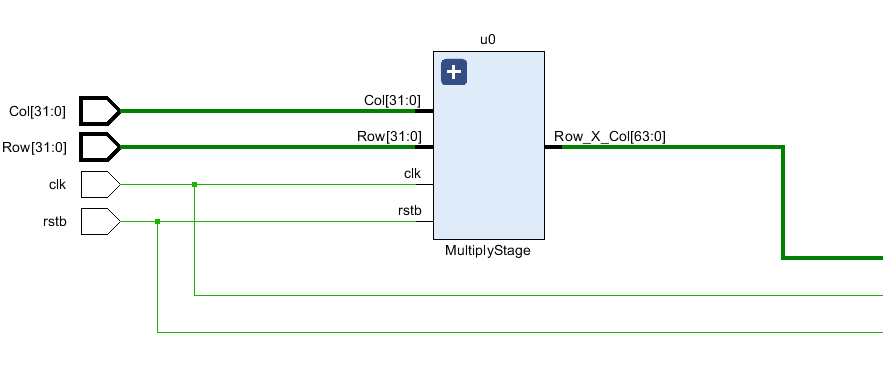
8. Redo part 4,5,6 for 8x8, 16x16 and 32x32 matrices.

**Schematics’ screenshots** are shown below:

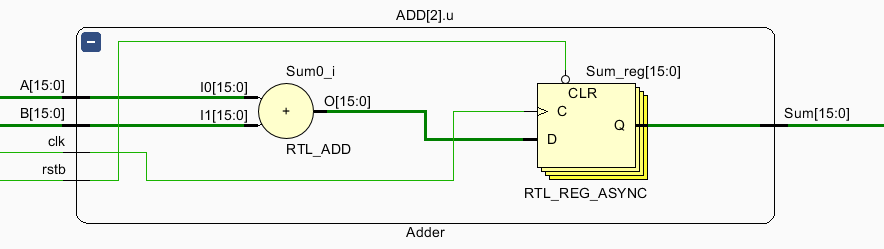
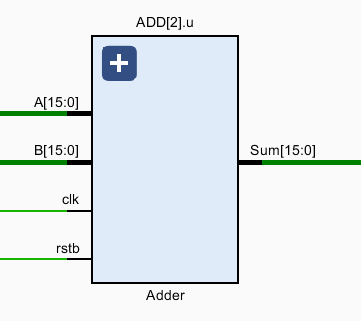
**MultiplyUnit** is designed to get the product of two 8-bit number A and B.

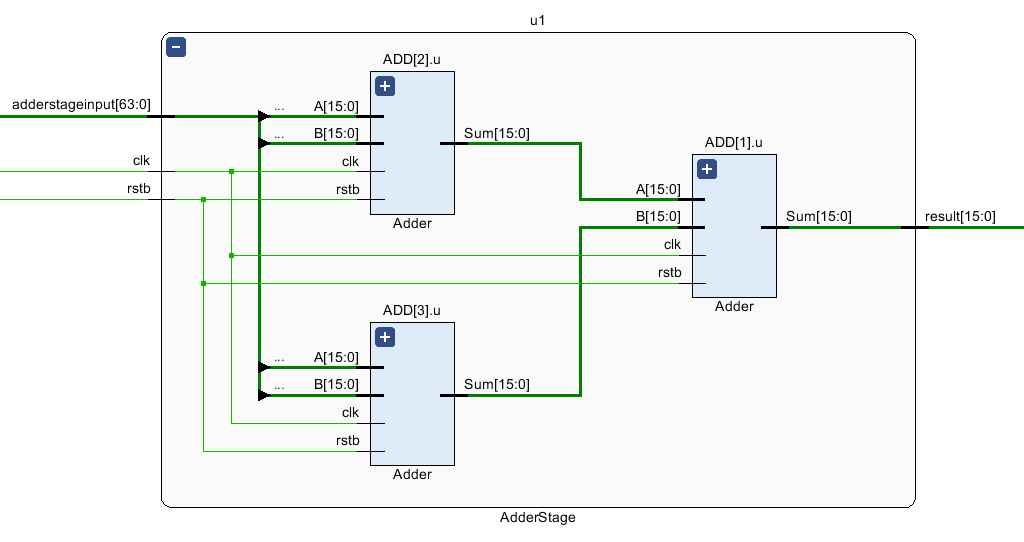
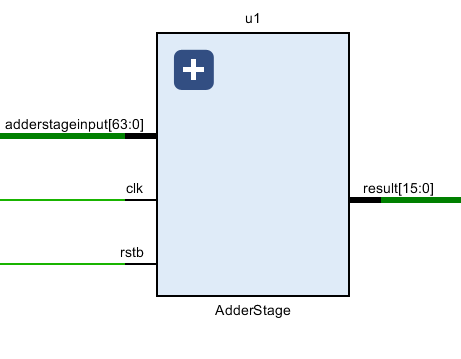
And according to the number of elements of matrix, combine all of the multiply units into **MultiplyStage**



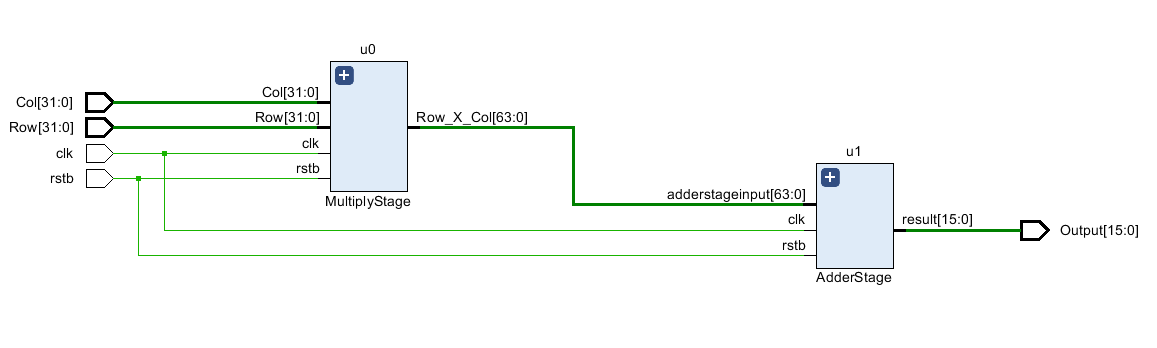
As for **Adder**, it computes the sum of two 16-bit numbers A and B.

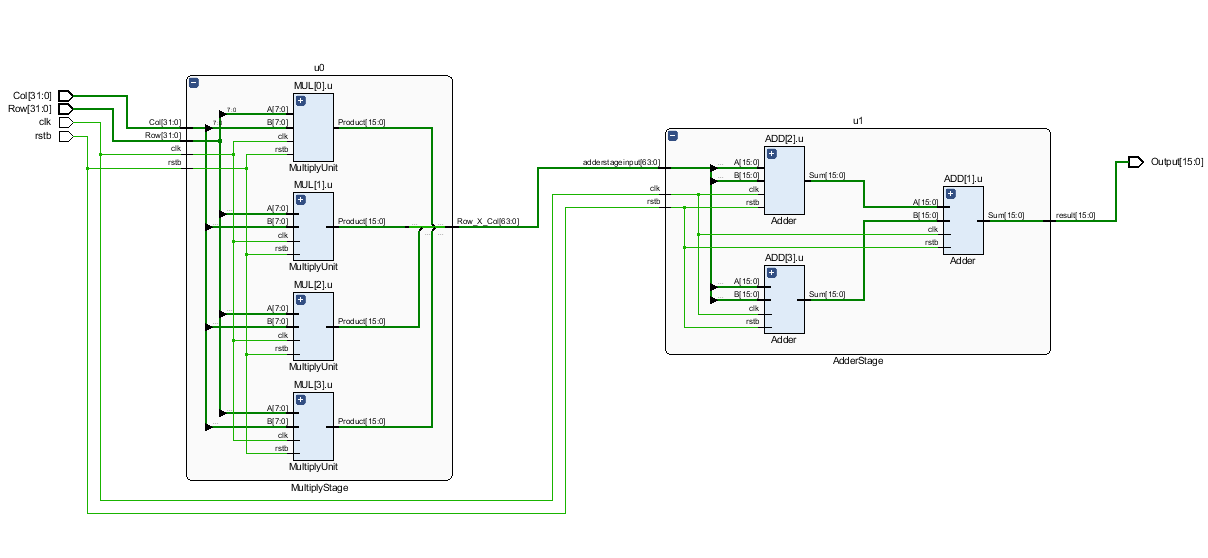


And **Adder**s make up the **AdderStage** module.



The Top Module, which is **MulandAddTree** consist of **MultiplyStage** and **AdderStage**.





// matrix1: Result:

// 1 2 3 4 1 2 3 4

// 2 3 4 5 2 3 4 5

// 3 4 5 6 3 4 5 6

// 4 5 6 7 4 5 6 7

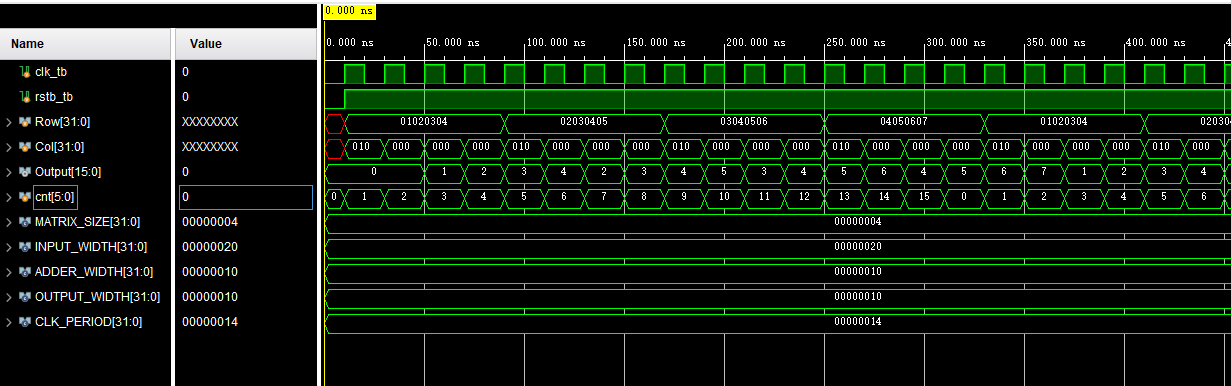
// matrix2:

// 1 0 0 0

// 0 1 0 0

// 0 0 1 0

// 0 0 0 1



// matrix3: Result:

// 1 2 3 4 30 40 50 60

// 2 3 4 5 40 54 68 82

// 3 4 5 6 50 68 86 104

// 4 5 6 7 60 82 104 126

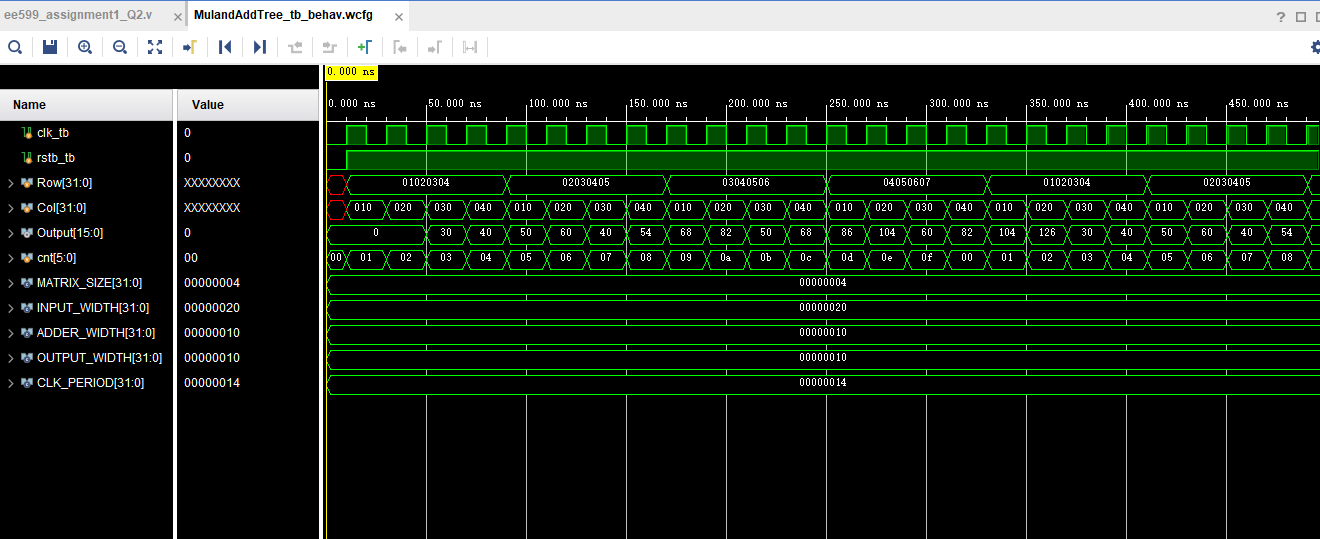
// matrix4:

// 1 2 3 4

// 2 3 4 5

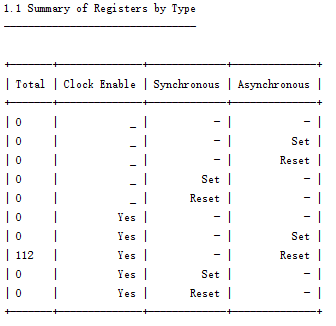
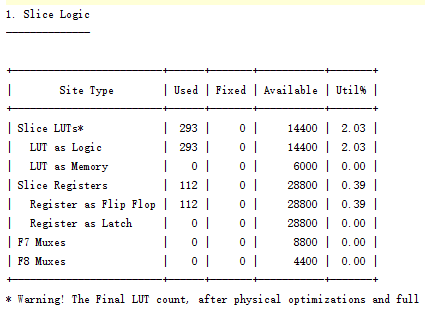
// 3 4 5 6

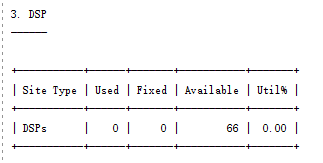
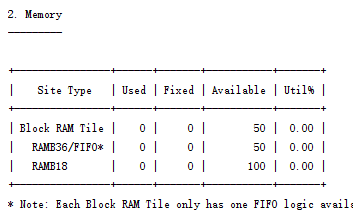
// 4 5 6 7

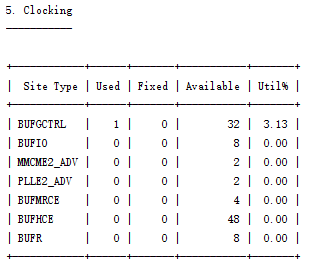
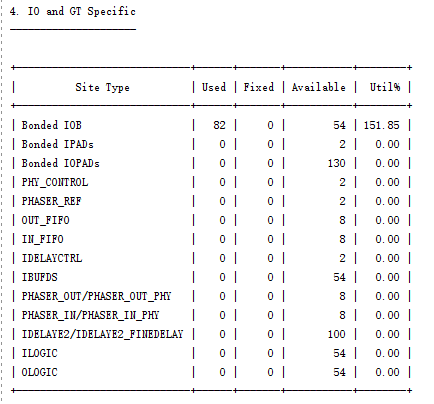


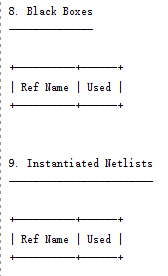
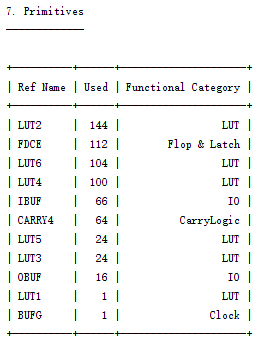
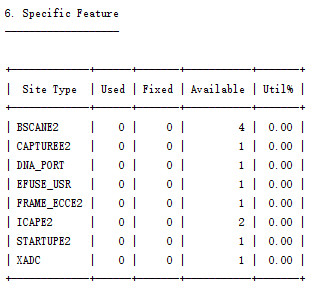
**The synthesis schematic please see in the Appendix.**

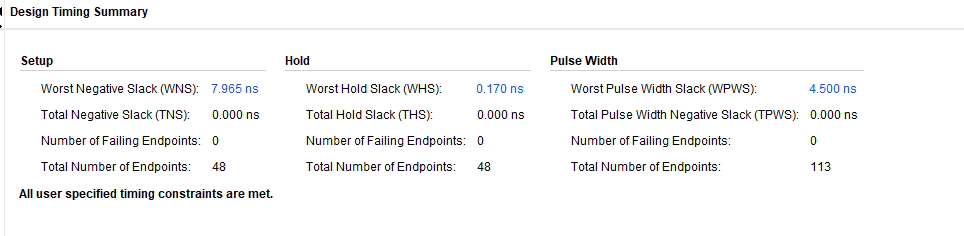
**Resource Estimation for 4x4 size:**

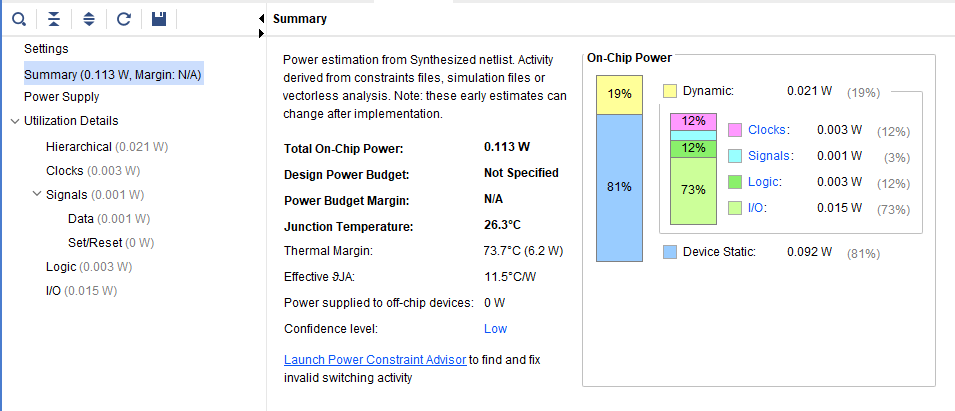




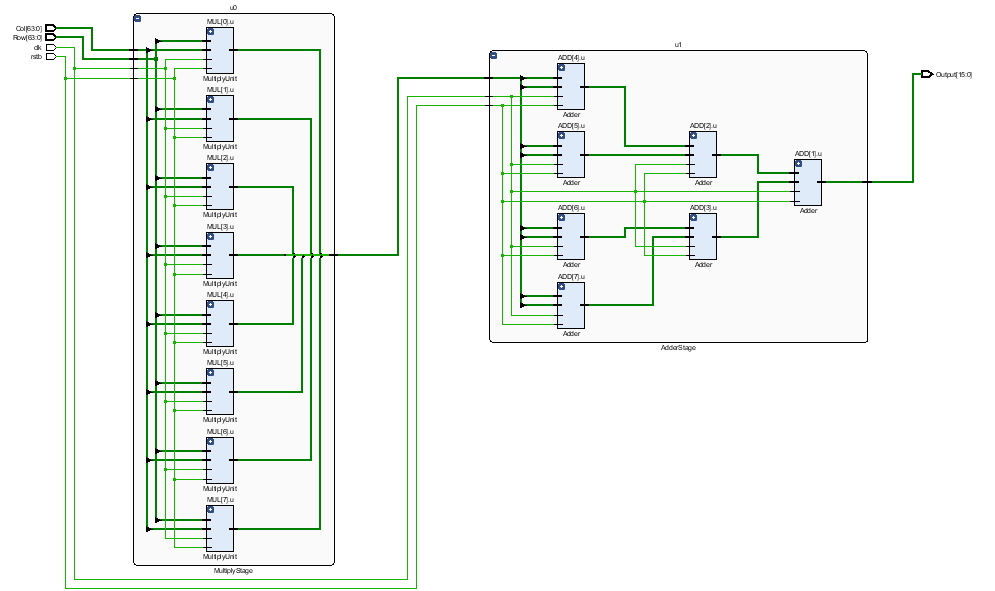






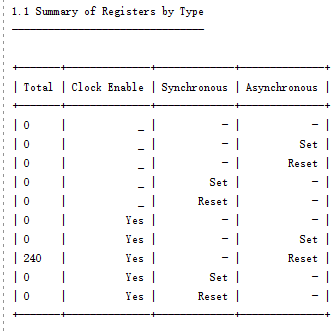
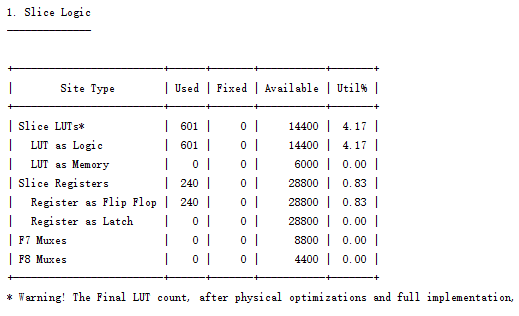


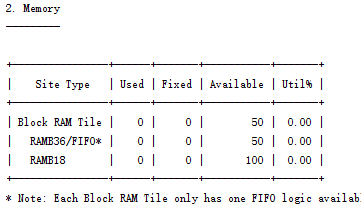
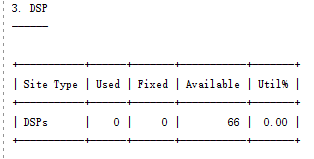
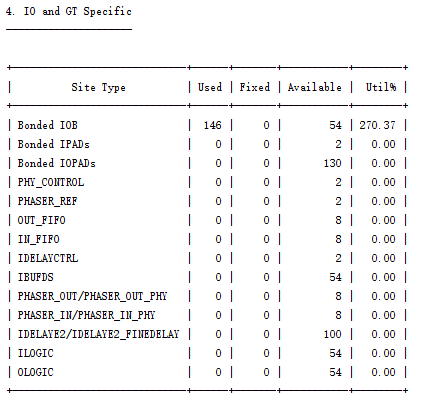
**Now redo for 8x8 size.**

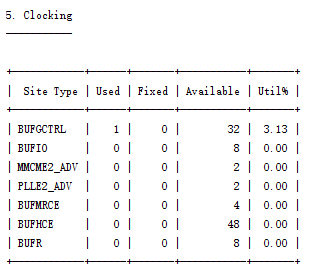
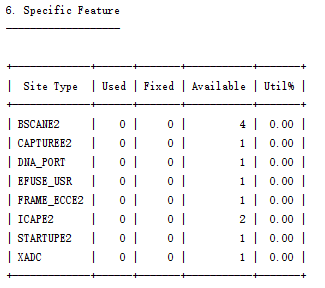
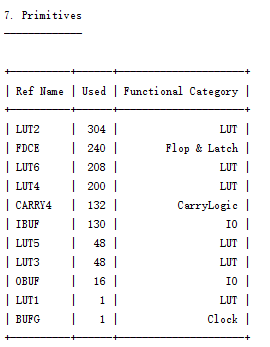


**The full synthesis schematic please see in the Appendix.**

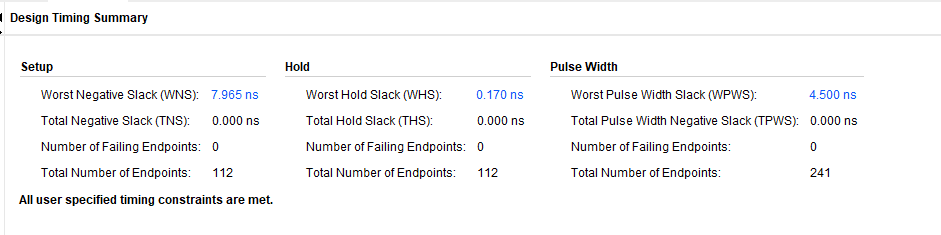
**Resource Estimation:**



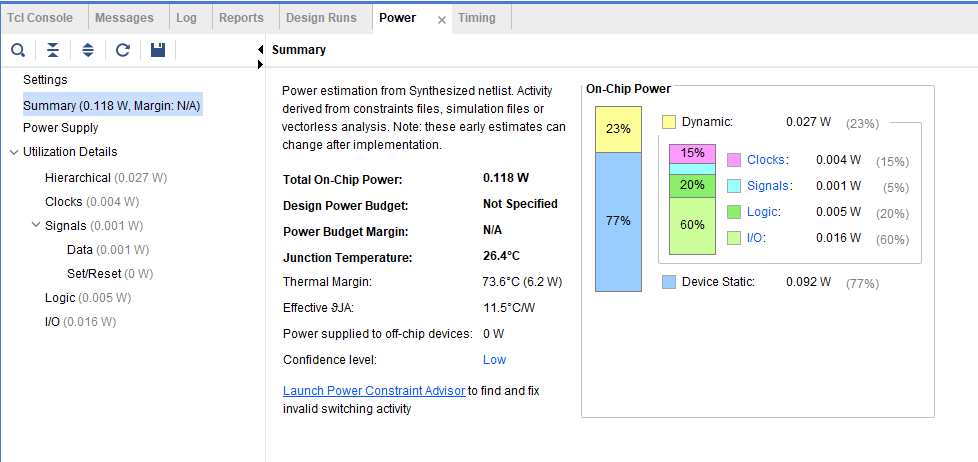
  

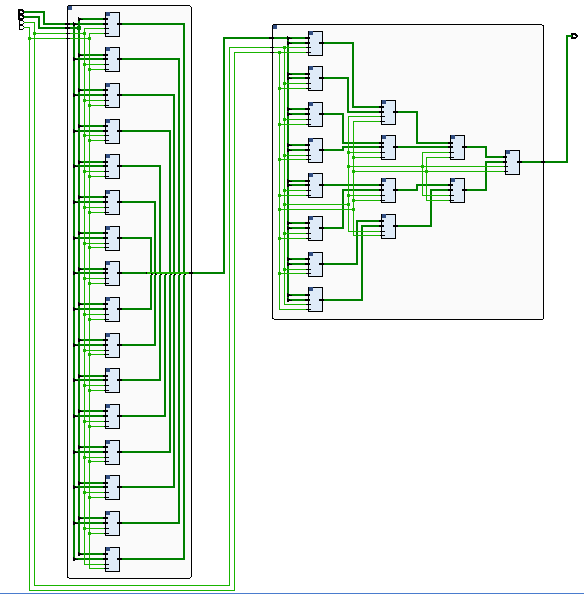
**Timing Estimation:**



**Power Estimation:**

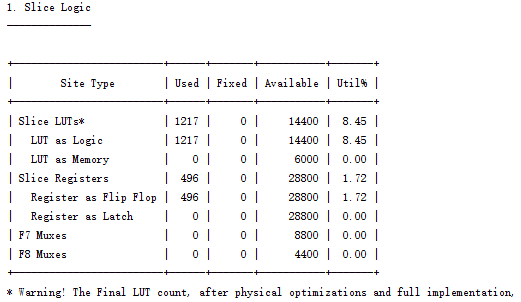
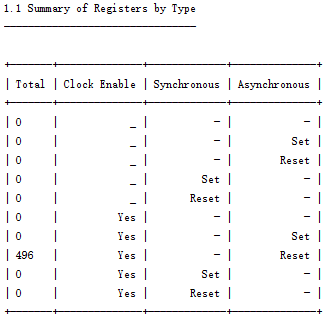
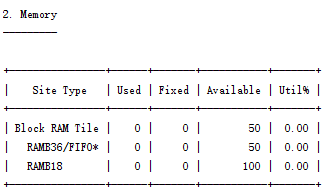
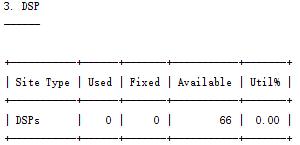


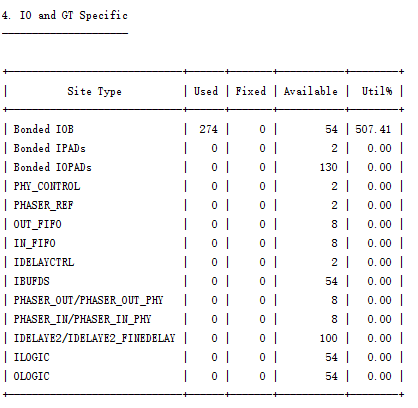
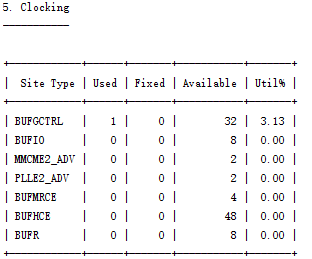
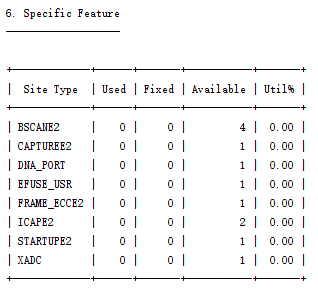
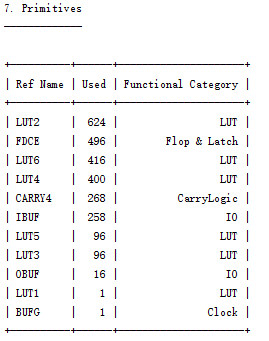
**Now redo for 16x16 size.**



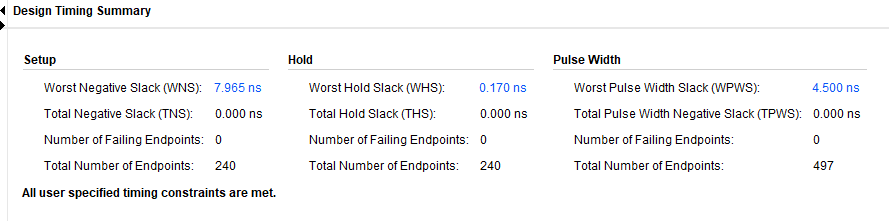
**The full synthesis schematic please see in the Appendix.**

**Resource Estimation:**

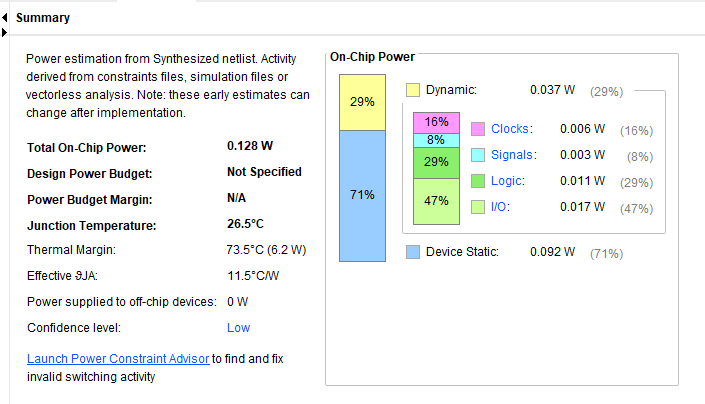
   

**Timing Estimation:**



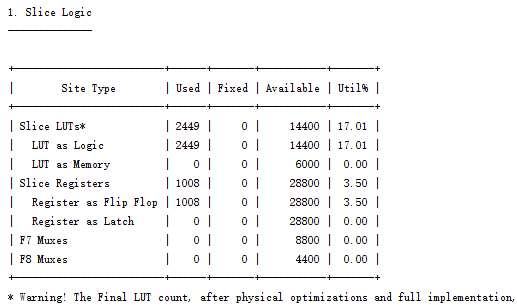
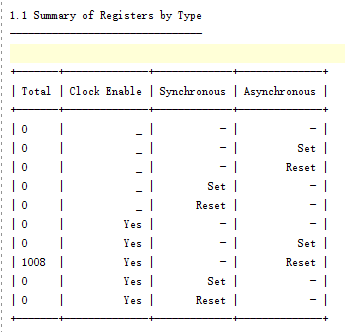
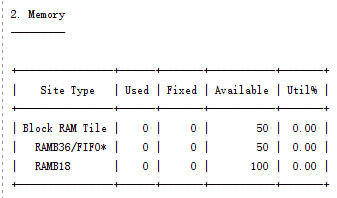
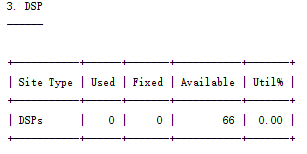
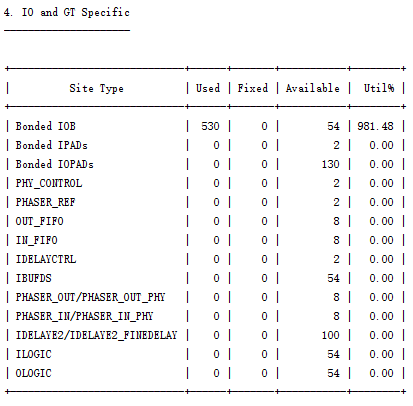
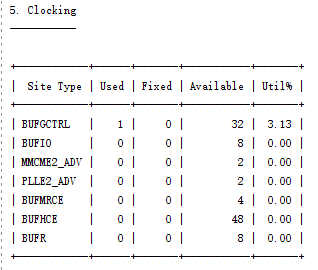
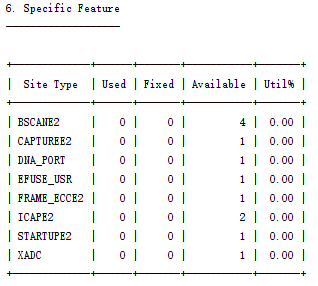
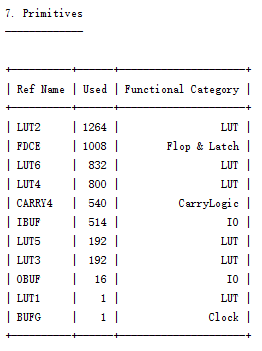
**Power Estimation:**



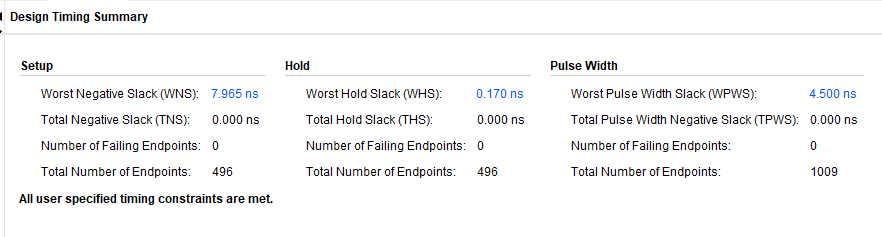
**Now redo for 32x32 size.**

**The full synthesis schematic please see in the Appendix.**

**Resource Estimation:**

**Timing Estimation:**



**Power Estimation:**

